Artificial and Computational Intelligence

AIMLCZG557

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Presented by
V Indumathi –Guest Faculty –BITS-WILP
indumathi.p@wilp.bits-pilani.ac.in

innovate achieve lead

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

AIMLCZG557 – CS#3

M1: Introduction to AI & Intelligent Agents & Environments

ACI: Disclaimer and Acknowledgements

⚠

- Few content for these slides may have been obtained from prescribed books and various other source(s) on the Internet.
- We hereby acknowledge all the contributors for their material and inputs and gratefully acknowledge people others who made their course materials freely available online. We have provided source information wherever necessary.
- We have added and modified the content to suit the requirements of the class dynamics & live session's lecture delivery flow for presentation.
- This *is not a full-fledged* reading materials. Students are requested to refer to the textbook w.r.t detailed content as per the course handout as shared over e-learning portal - Taxila.
- **Slide Source / Preparation / Review:**
- From BITS Pilani WILP**:** Prof. Rajavadhana, Prof. Indumathi, Prof. Sangeetha and Prof Parthasarathy PD
- From BITS On-campus & External : Mr.Santosh GSK

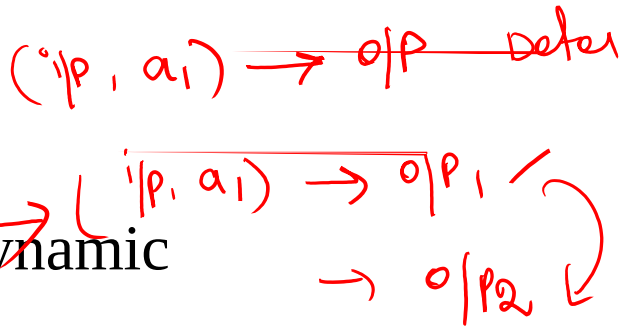
Agenda for CS #3

1) Recap of CS #2
2) Problem Formulation
3) Uninformed Search algorithms
    1) BFS, DFS, UCS
    2) Iterative Deepening Search
4) Informed Search algorithms
4) Q&A!

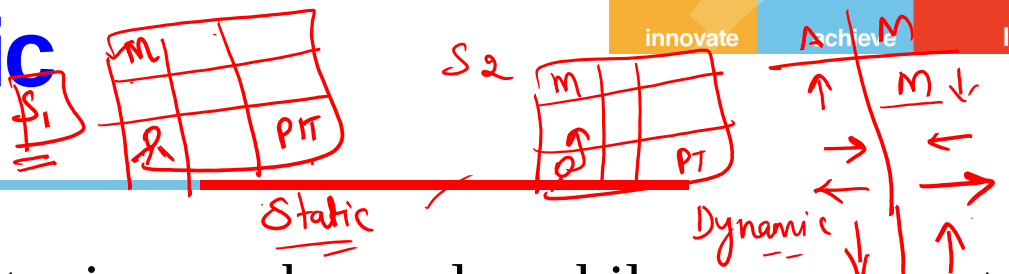
Agent, Envi, sensor, Actuati

Properties of Environments

- Observability: full *vs.* partial *vs. non*
- Deterministic *vs.* stochastic
- Episodic *vs.* sequential
- Static *vs. Semi-dynamic vs.* dynamic
- Discrete *vs.* continuous
- Single Agent *vs.* Multi Agent (Cooperative, Competitive, Self-Interested)



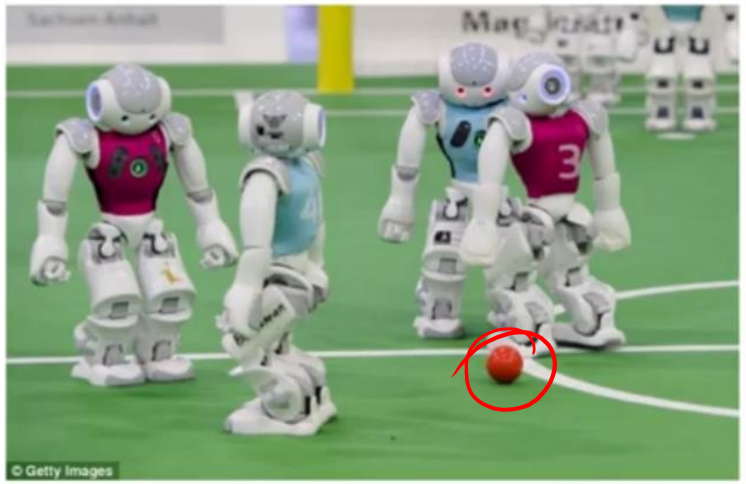
Static vs Dynamic



- Static: The environment is unchanged while an agent is deliberating/thinking. Ex: Puzzle, rubik's cube etc.
- Dynamic: Environment can change while agent is thinking. Ex: raining, signals etc.
- Semidynamic: Environment does not change with time, but performance scored does. Ex: chess with timing…

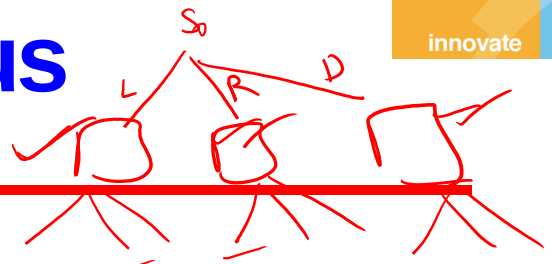
Rubik's Cube

VS.



Football playing robots

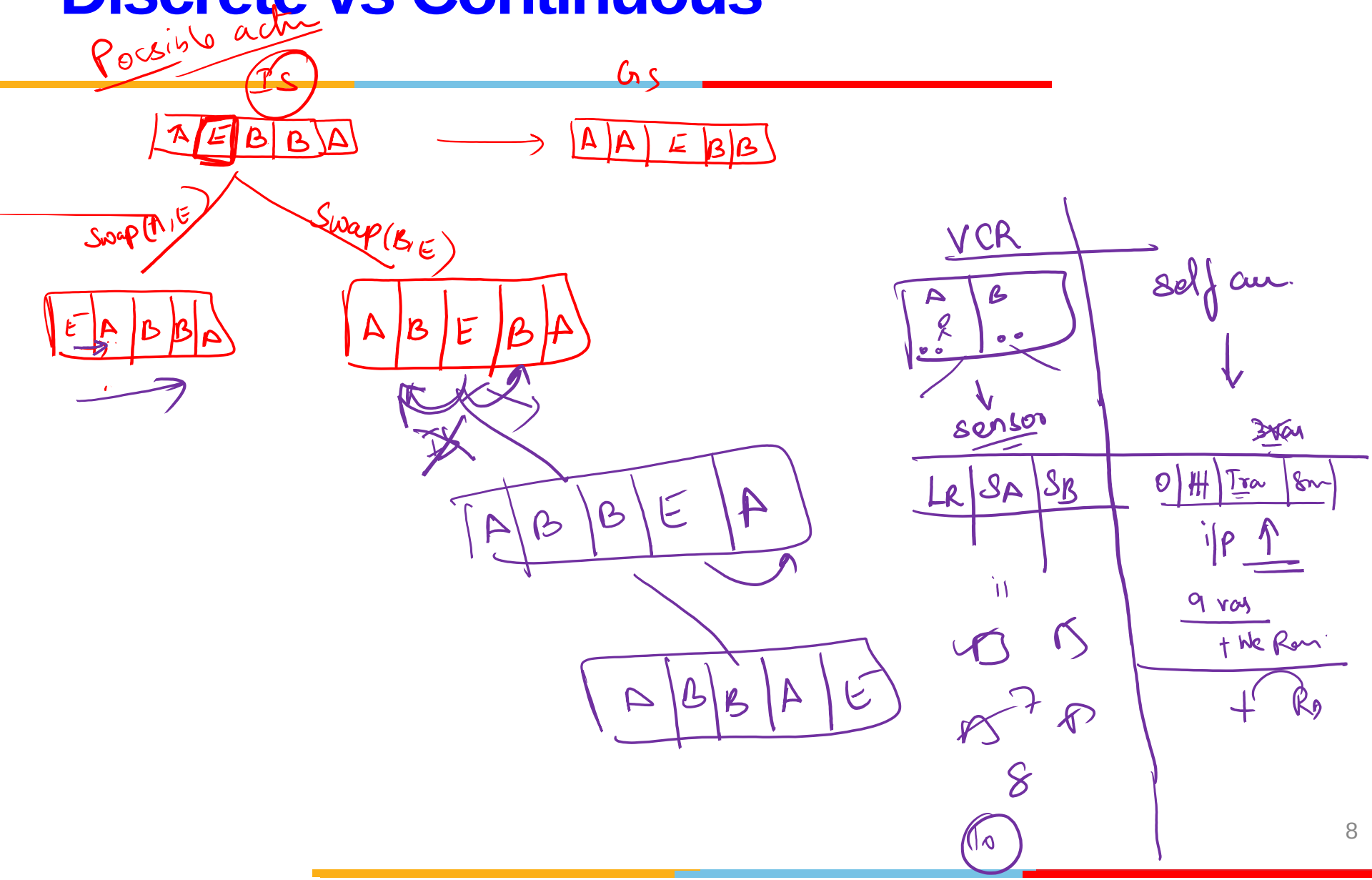
Discrete vs Continuous



- Discrete : A limited number of distinct, clearly defined percepts and actions. Ex: chess game
- Continuous environments have a certain level of mismatch with computer systems. Ex: taxi driving.

VS.

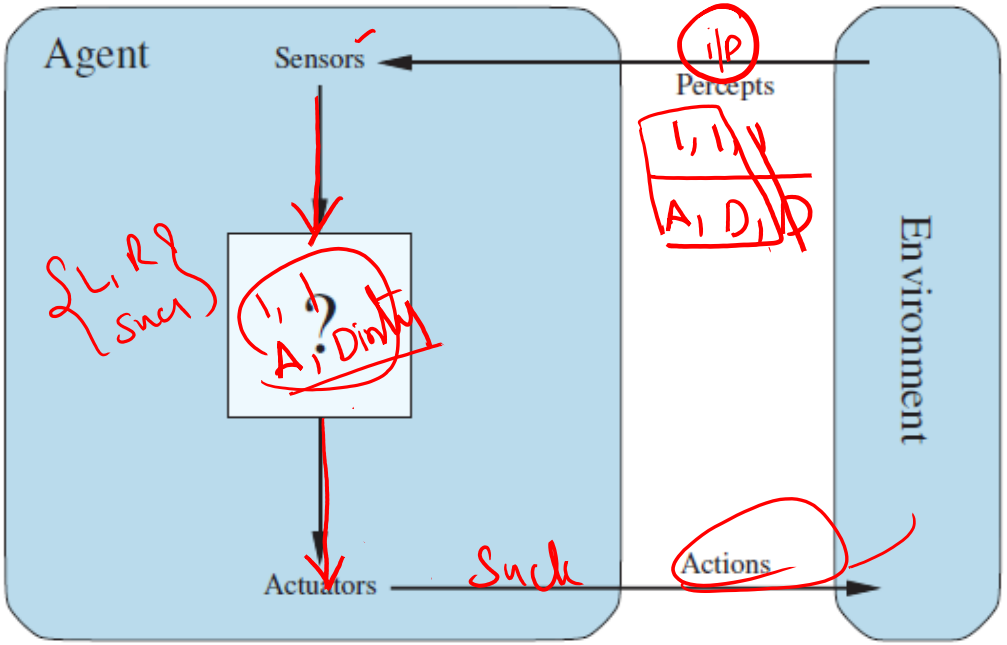
Discrete vs Continuous



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Agent functions and programs

- An agent is completely specified by the agent function mapping percept sequences to actions



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Table-lookup agent

function TABLE-DRIVEN-AGENT(*percept*) **returns** an action

persistent: *percepts*, a sequence, initially empty

table, a table of actions, indexed by percept sequences, initially fully specified

append *percept* to the end of *percepts*

action ←LOOKUP(*percepts*, *table*)

return *action*

Percept sequence	Action
[*A*, *Clean*]	*Right*
[*A*, *Dirty*]	*Suck*
[*B*, *Clean*]	*Left*
[*B*, *Dirty*]	*Suck*
[*A*, *Clean*], [*A*, *Clean*]	*Right*
[*A*, *Clean*], [*A*, *Dirty*]	*Suck*
⋮	⋮
[*A*, *Clean*], [*A*, *Clean*], [*A*, *Clean*]	*Right*
[*A*, *Clean*], [*A*, *Clean*], [*A*, *Dirty*]	*Suck*
⋮	⋮

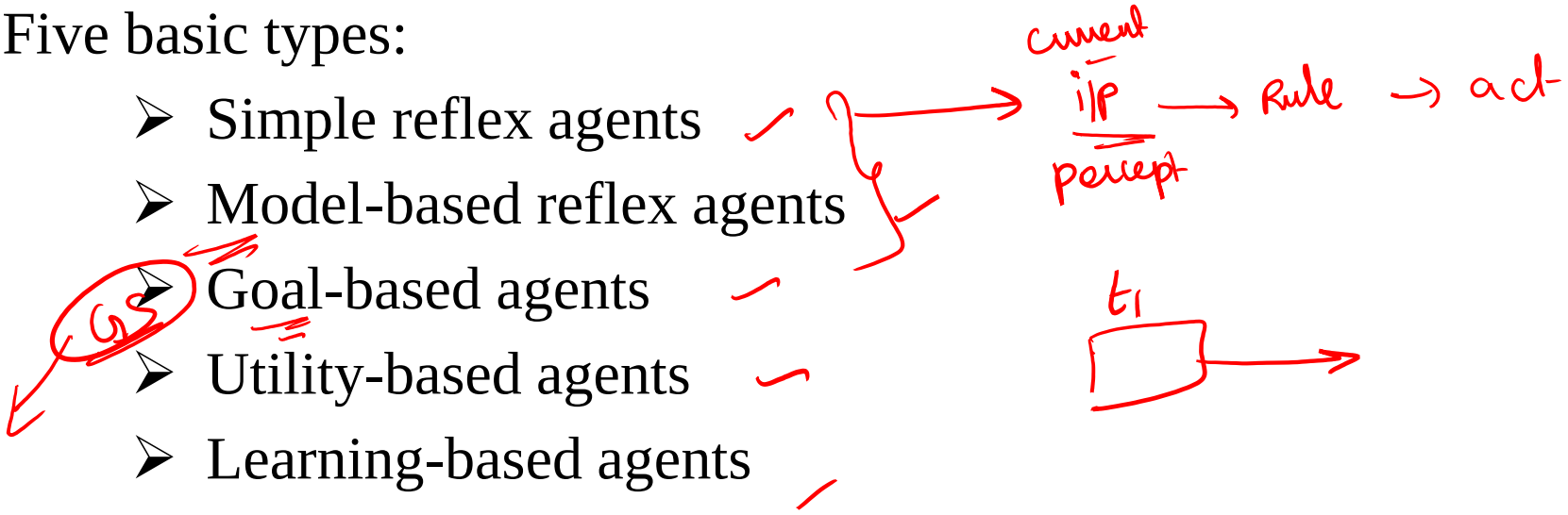
Drawbacks:

- Huge table
- Take a long time to build the table
- No autonomy
- Even with learning, need a long time to learn the table entries

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Agent types

- Five basic types:
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
  - Utility-based agents
  - Learning-based agents

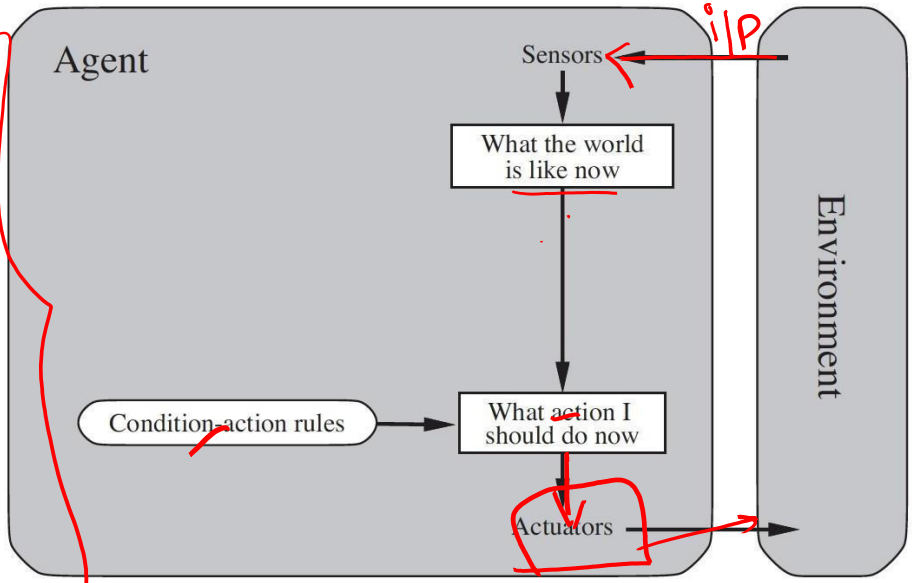


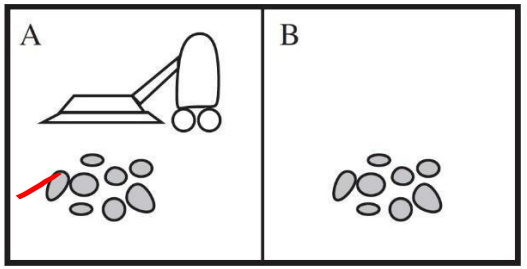
BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Agent Architectures: Simple Reflex Agent

```
function SIMPLE-REFLEX-AGENT(percept) returns an action
    persistent: rules, a set of condition–action rules
    state ← INTERPRET-INPUT(percept)
    rule ← RULE-MATCH(state, rules)
    action ← rule.ACTION
    return action
```

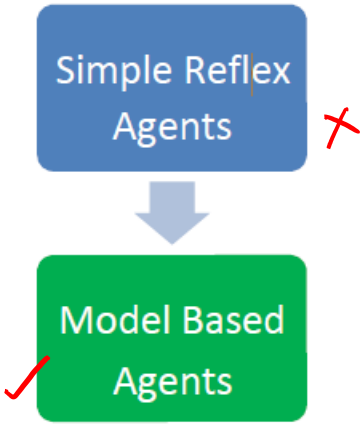
```
function REFLEX-VACUUM-AGENT( [location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

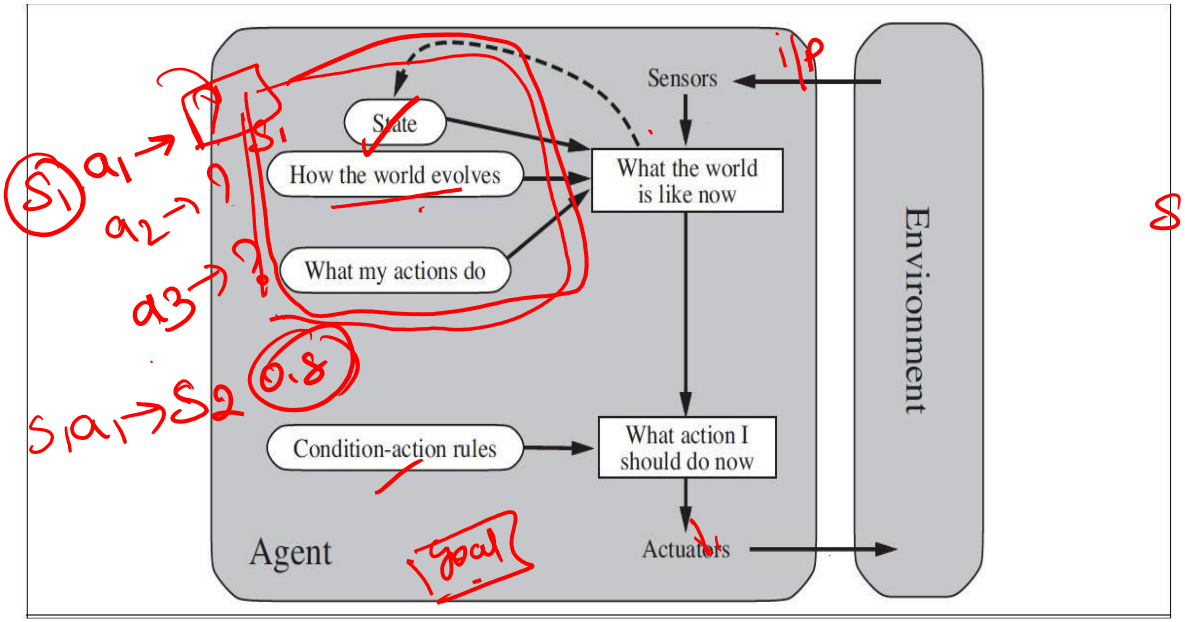




BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Agent Architectures: Model Based Agent





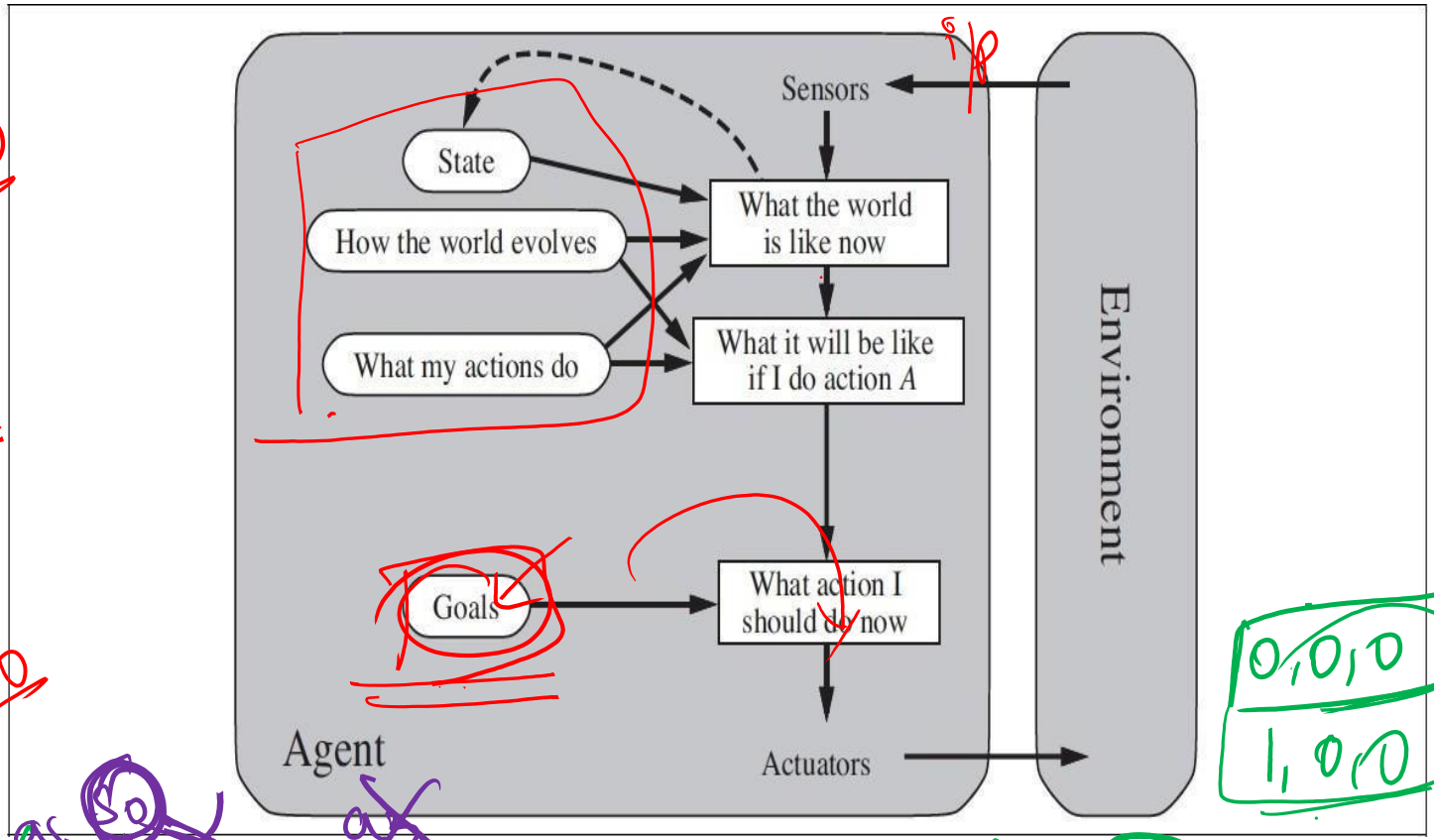
function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action

persistent: *state*, the agent's current conception of the world state
transition model, a description of how the next state depends on the current state and action
sensor model , a description of how the current world state is reflected in the agent's percepts
rules, a set of condition–action rules
action, the most recent action, initially none

state ← UPDATE-STATE(*state, action, percept, transition model, sensor model*)
rule ← RULE-MATCH(*state, rules*)
action ← *rule*.ACTION
return *action*

Agent Architectures: Goal Based Agent

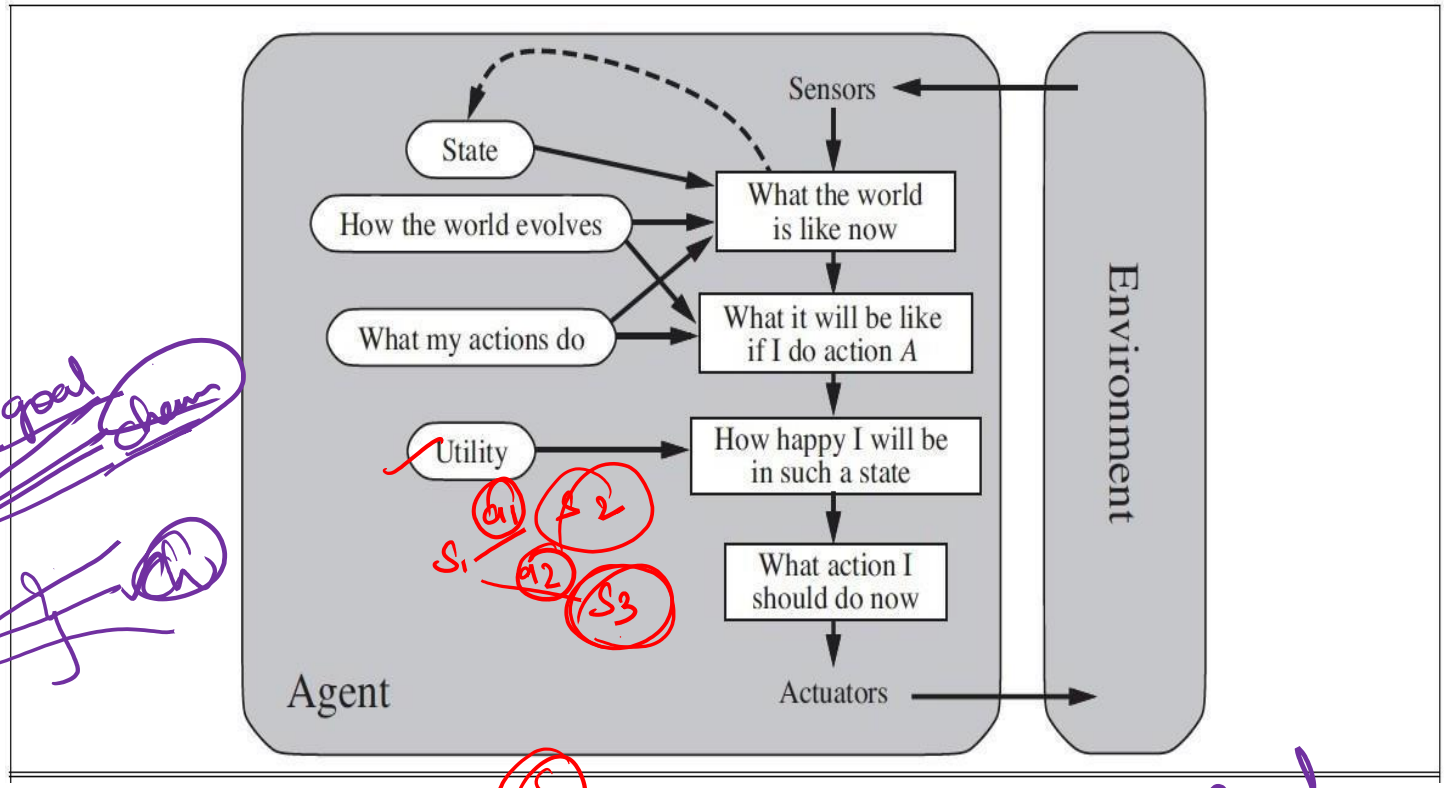
- Simple Reflex Agents
- Model Based Agents
- Goal Based Agents



Agent Architectures: Utility Based Agent

innovate achieve lead

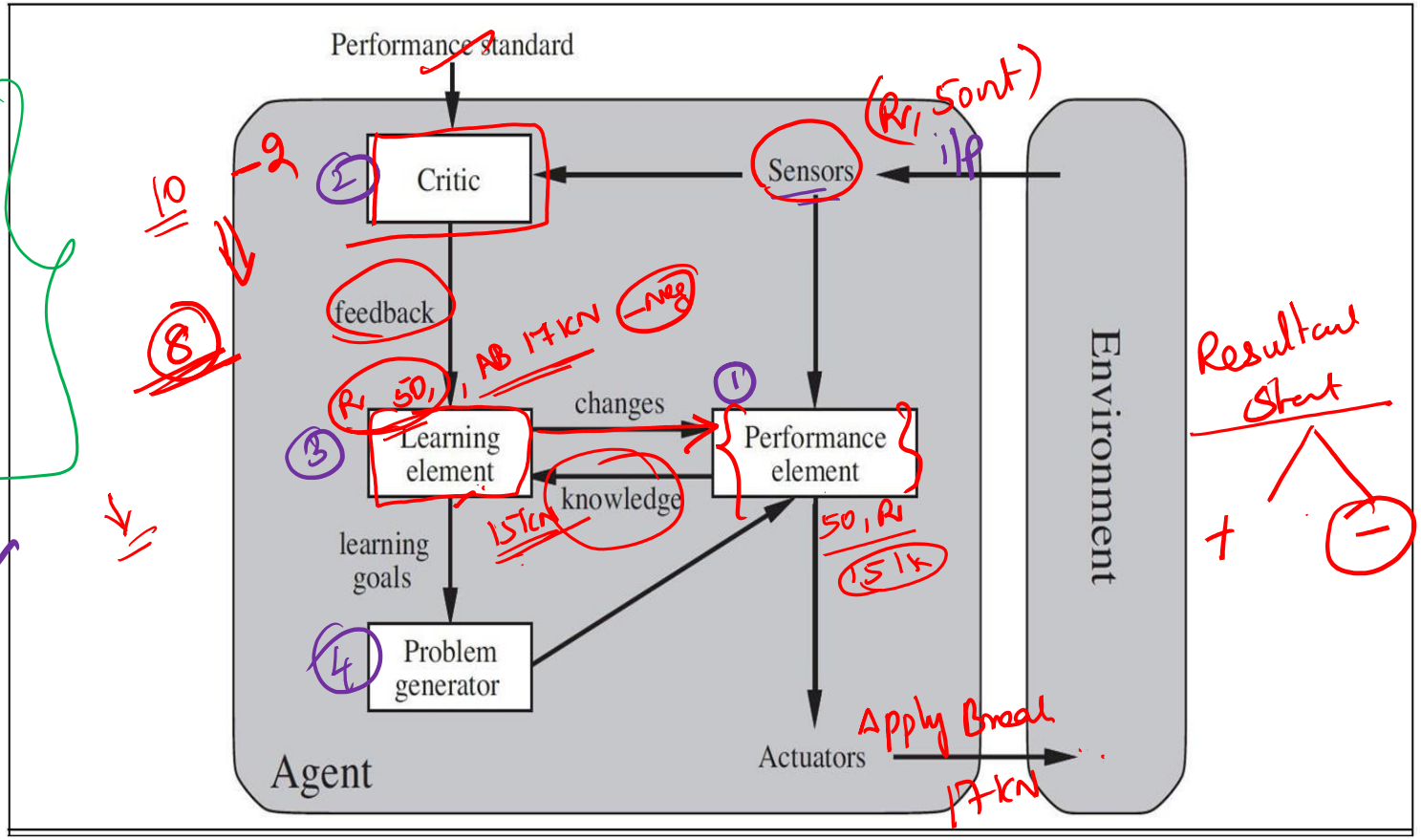
- Simple Reflex Agents
- Model Based Agents
- Goal Based Agents
- Utility Based Agents



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

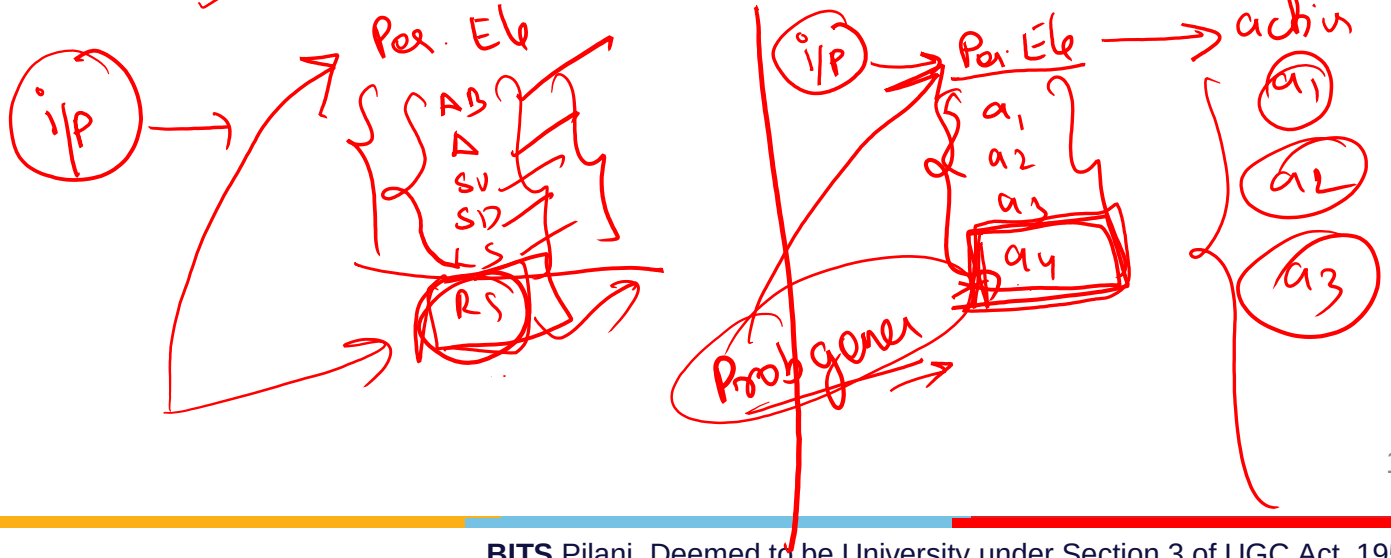
Agent Architectures: Learning Based Agent

- Simple Reflex Agents
- Model Based Agents
- Goal Based Agents
- Utility Based Agents
- Learning Agents



Role of Learning

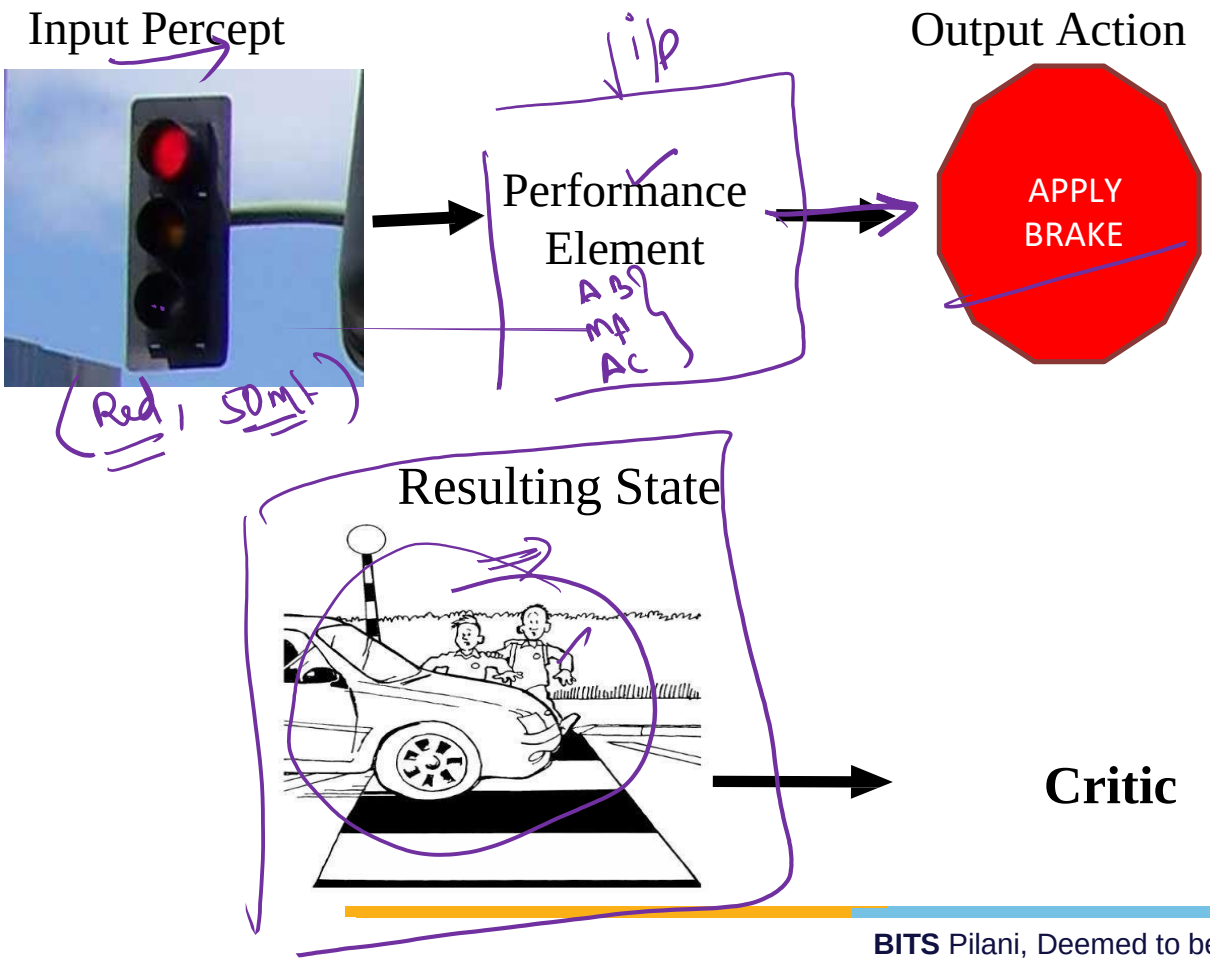
- **Performance Element** – taking a decision of action based on percepts
- **Learning Element** – Make the performance element select better actions such that the utility function is optimized
- **Critic** – Provides feedback on the actions taken
- **Problem Generator** – Make the Performance Element select sub-optimal actions such that you would learn from unseen actions



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

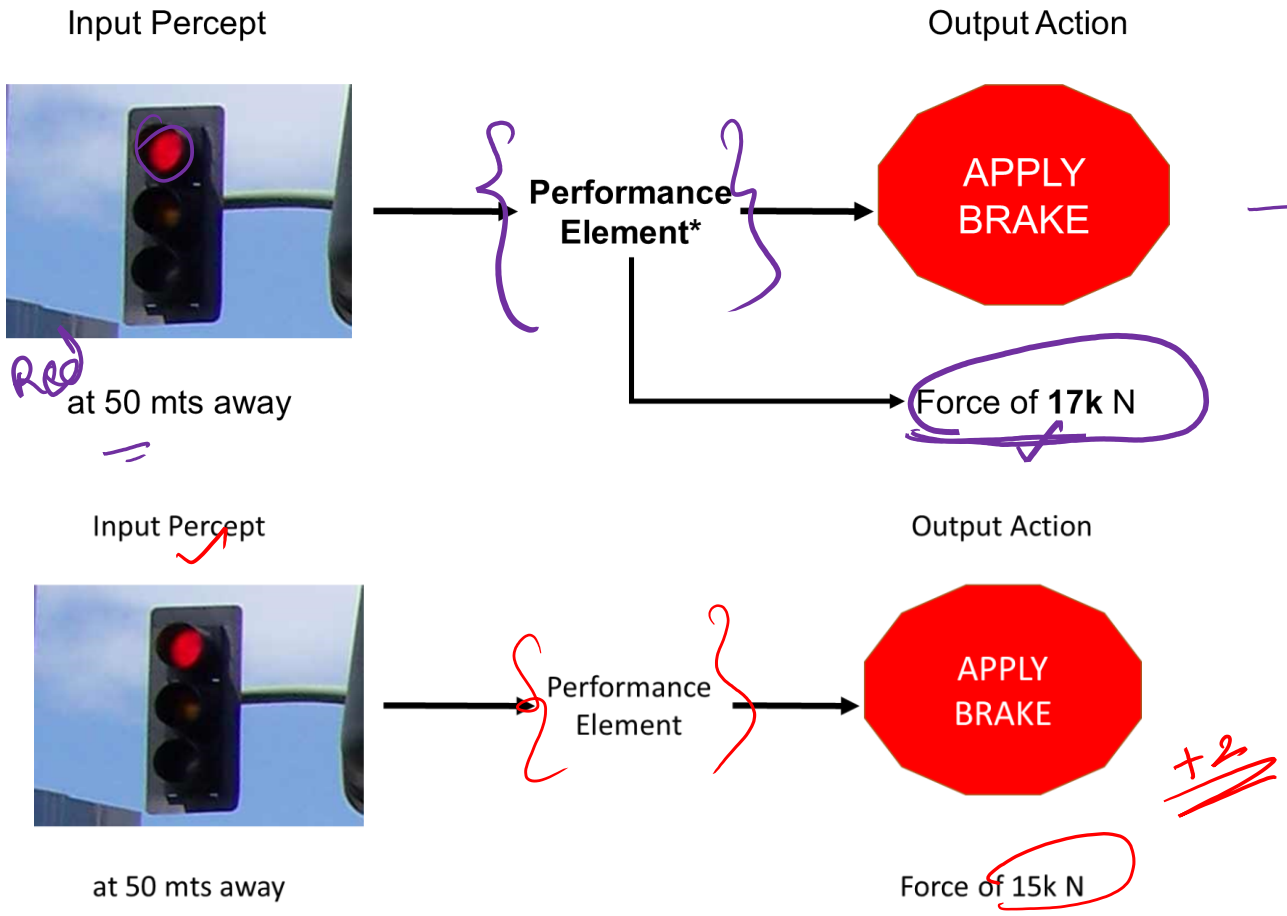
Role of Learning

Agents that improve their performance by learning from their own experiences



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Role of Learning



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Problem Solving Agents

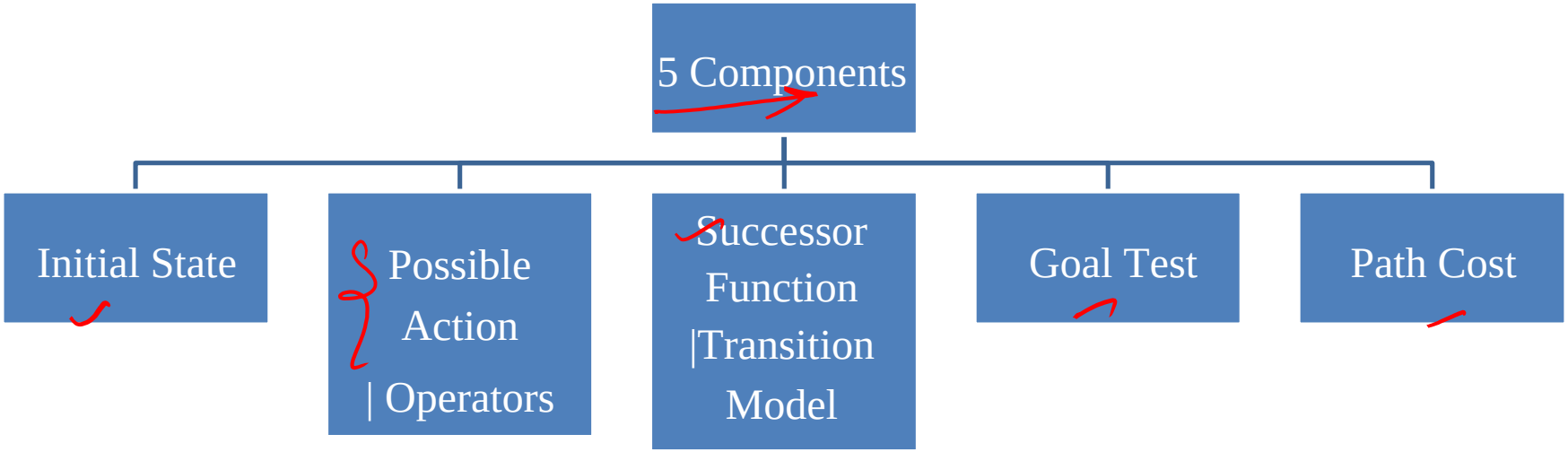
1. Formulate – Define the problem and its components
2. Search – Solve the problem using search algorithms
3. Execute – Execute the solution of fixed sequence of actions

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Problem Solving Agents - Problem Formulation

Abstraction Representation:
Decide what actions under states to take to achieve a goal



A function that assigns a numeric cost to each path. A path is a series of actions. Each action is given a cost depending on the problem.

Solution = Path Cost Function + Optimal Solution

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

What is a State ?

- All information about the environment
- All information necessary to make a decision for the task at hand.

What is a state space ?

- A State space is the set of all states reachable from the initial state.
- A state space forms a graph in which the nodes are states and the arcs between nodes are actions.
- In the state space, a *path* is a sequence of states connected by a sequence of actions.
- The solution of a problem is part of the graph formed by the state space.
- The state space representation forms the basis of most of the AI methods.

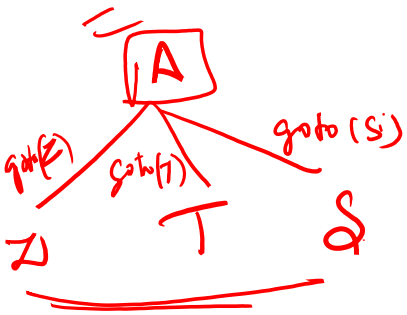
Problem Solving Agents

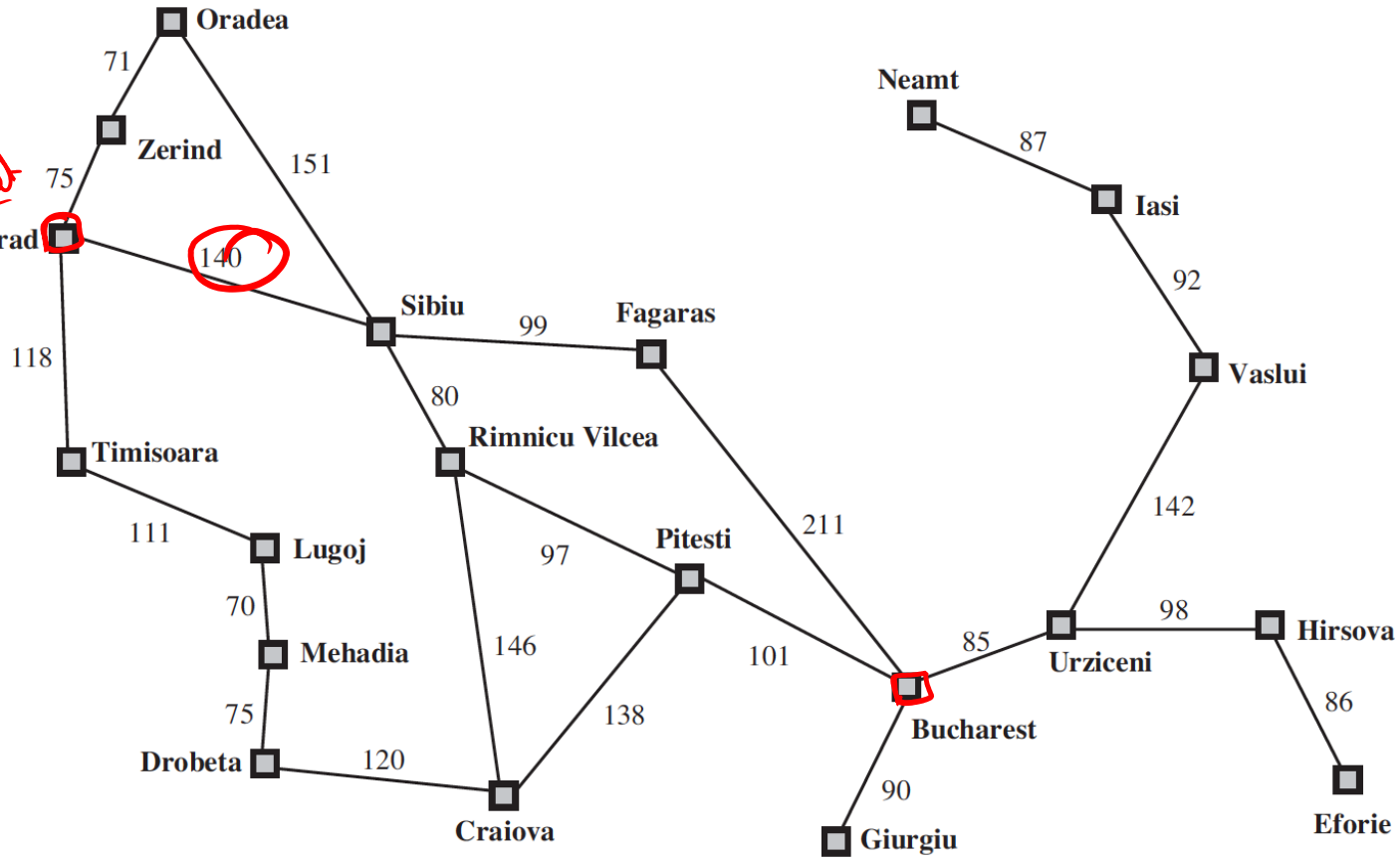
- **Problem Formulation**: Process of deciding what actions and states to consider, given a goal …
- Unknowns about the Environment
  - Doesn't know what state the agent is in
  - Estimation of number of possible actions
  - Doesn't know the value of actions
  - Doesn't know the resulting state of actions
- Assumptions about the Environment
  - Observable: Agent always knows the current state
  - Discrete: At any given state, only finitely many actions to choose from
  - Known: Knows which states are reached by each action
  - Deterministic: Each action would always have the same resulting state

- Under these assumptions, the solution would be
- *A Fixed Sequence of Actions*
- **Search**: The process of looking for a fixed sequence of actions that reaches a goal
- **Search Algorithm**: Takes problem as input and outputs sequence of actions

Problem Solving Agents

Agent is here





BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Problem Solving Agents – Formulate

A problem can be formulated using five components

- **Initial State –** The agent's starting state. E.g., *In(Arad)*
- **Possible Actions –** Set of applicable actions in a given state. E.g., from the state *s = In(Arad)*, applicable actions

  `ACTIONS(s)` ➔ `{Go(Sibiu), Go(Timisoara), Go(Zerind)}`
- **Transition Model –** The resulting state of an action in a given state, modeled as RESULT(s, a)

  `RESULT( In(Arad), Go(Sibiu) ) = In(Sibiu)`
- **Goal Test –** Determines whether a given state is a goal state.

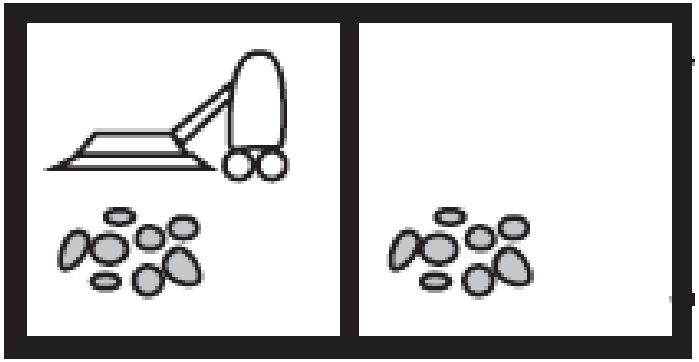
  `IsGoal( In(Bucharest) ) = Yes`
- **Path Cost –** A function that assigns a numeric cost to each path. A path is a series of actions. Each action is given a cost depending on the problem.

  `cost( In(Arad), go(Sibiu)) = 140 kms`

Problem Solving Agents – Formulate

- In the above definition
  - Many possible states – In(Arad), the agent has many possibilities in real world, e.g., can be in the middle of Arad, near an airport, shopping center, hotel, etc.
  - Many possible actions – Agent can take several actions while driving from Arad to Sibiu. It can halt at a place, take a detour, turn the radio on.
  - However, they are irrelevant for finding the solution of path to Bucharest
  - Hence, can be abstracted

Formulate – Toy Problem Example

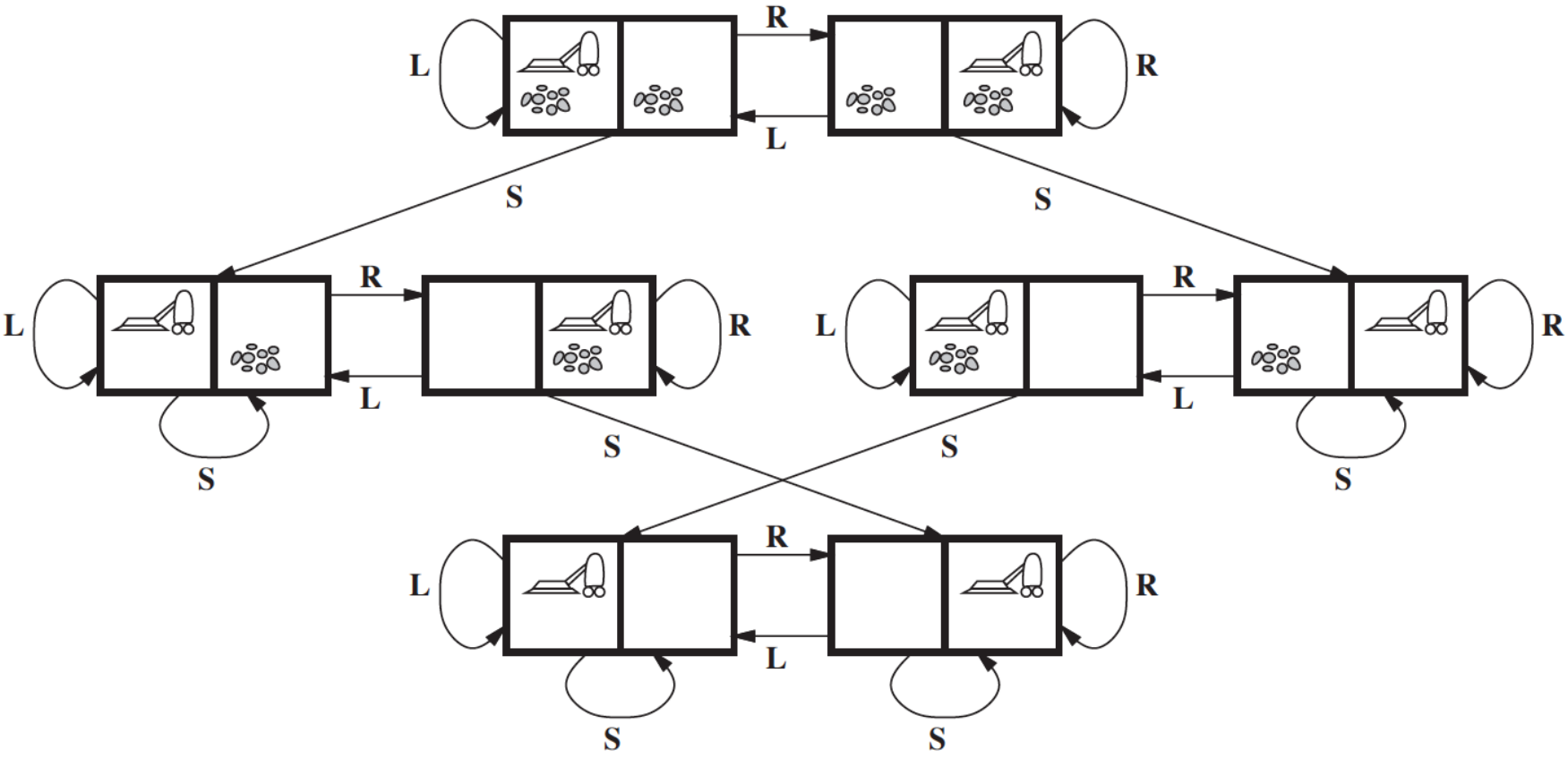


Vacuum World

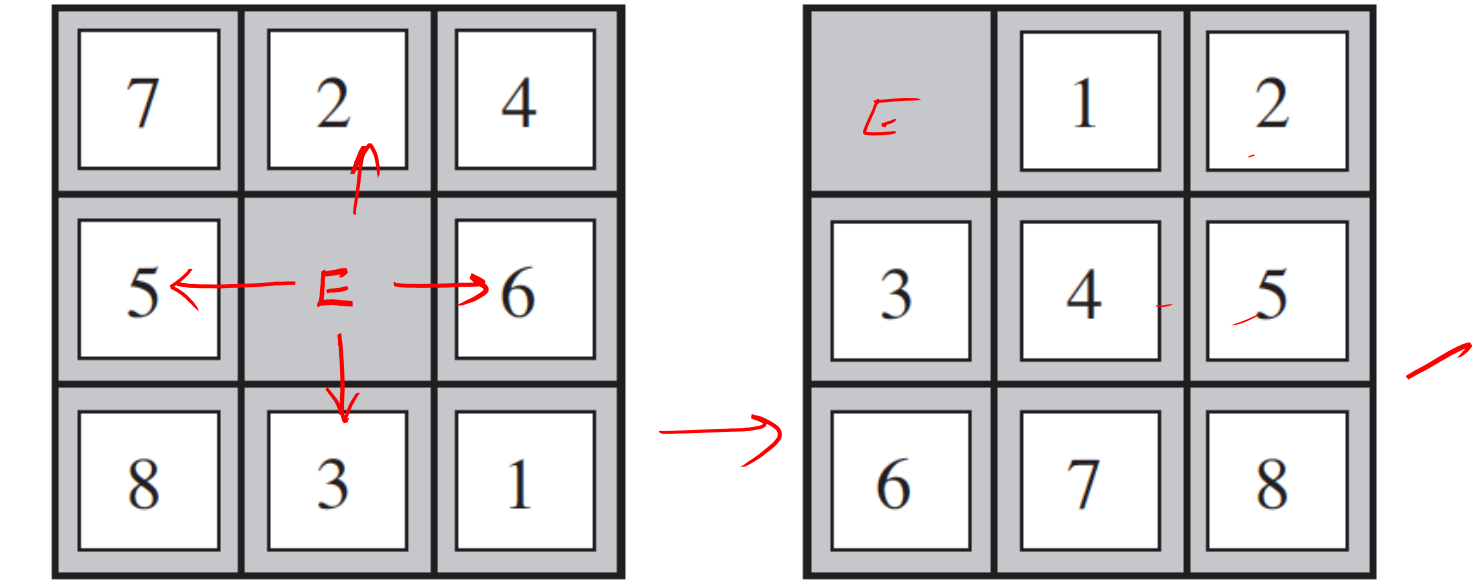
Possible States – There are two locations and two possibilities of dirt. Hence, there would be a total of 8 possible states, i.e., $2x2^2$

1. Initial State – Any state with position of agent and dirt mentioned
2. Actions – Three possible actions, Left, Right, Suck
3. Transition Model – Left would move the agent to left location, except when the agent is in leftmost location, there would be no effect. Similarly for Right action. Suck would remove the dirt, if any.
4. Goal Test – Checks whether all locations are clean
5. Path cost – Each step costs 1, so the path cost is total number of steps

Formulate – Toy Problem Example



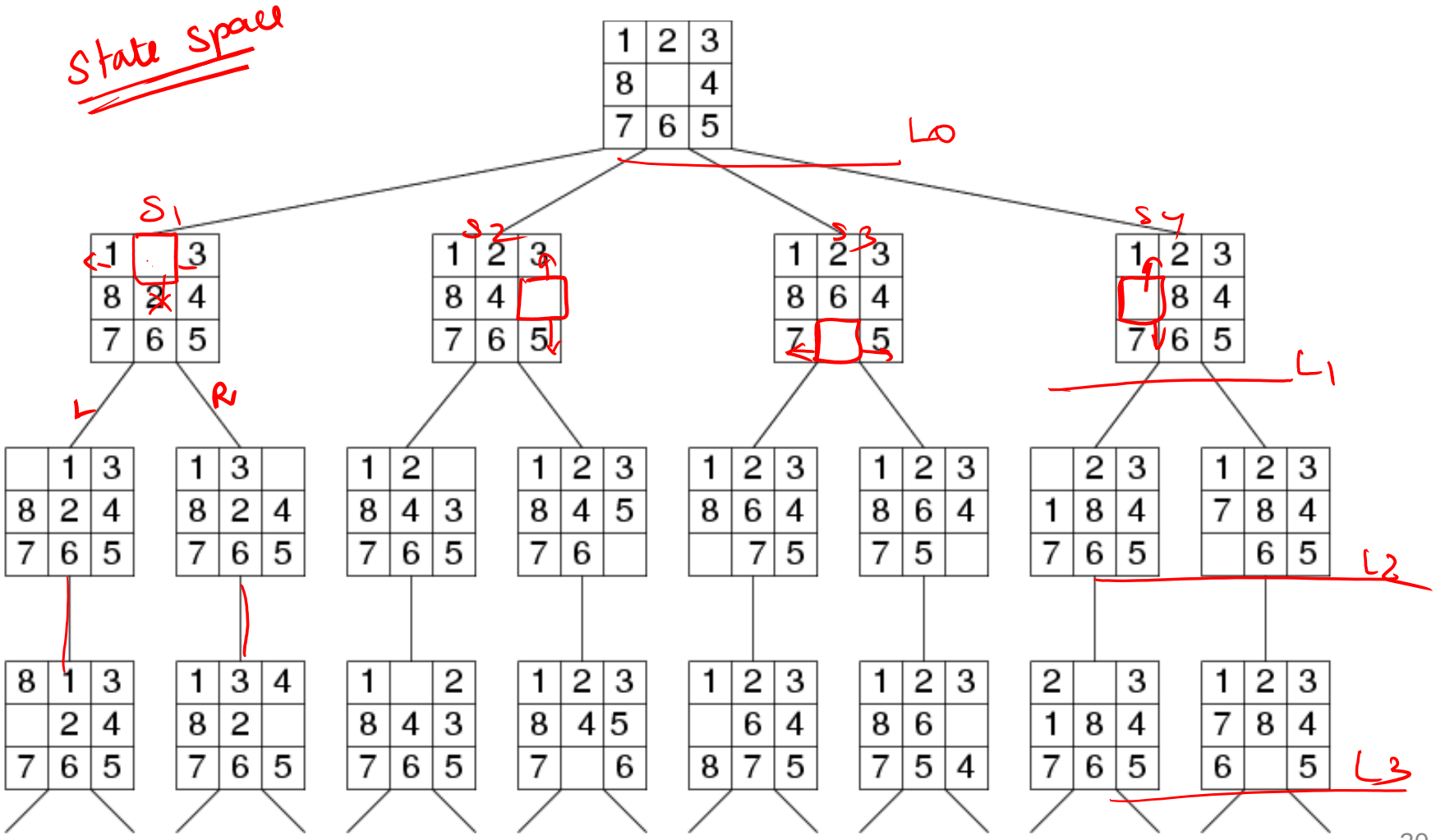
Formulate – Toy Problem Example



8-puzzle

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

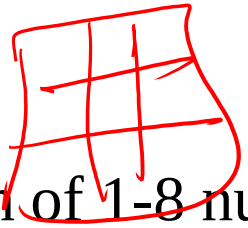
Fragment of 8-Puzzle Problem Space



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Formulate – Toy Problem Example

Possible states = 9! / 2

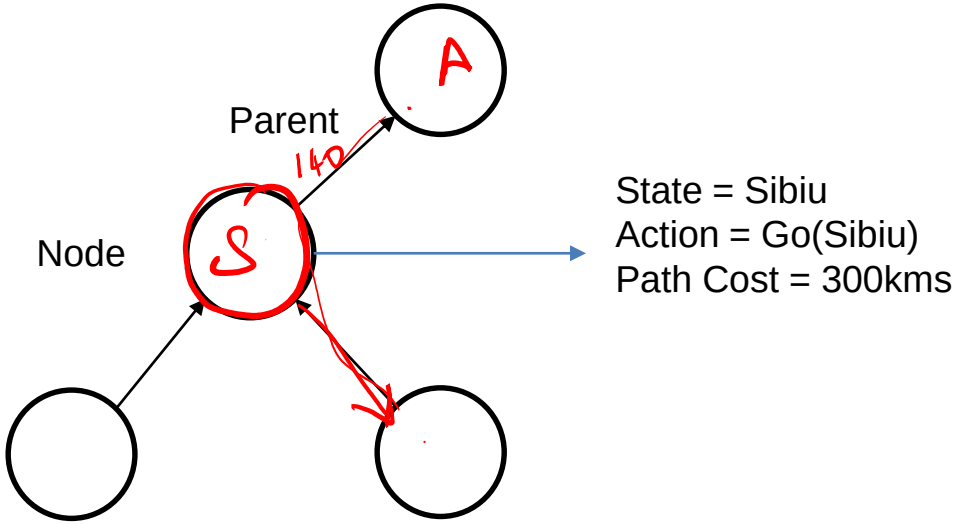


1. Initial State – Any permutation of 1-8 numbers with a blank
2. Actions – Movement of blank space either by Left, Right, Up or Down
3. Transition Model – The resulting state after moving the blank space will replace the digit
4. Goal Test – Check whether the state matches the goal configuration
5. Path cost – Each step costs 1, so the path cost is total number of steps

Implementation: states vs. nodes

For each node *n* of the tree,

- *n.STATE:* the state in the state space to which node corresponds
- *n.PARENT:* the node in the search tree that generated this node
- *n.ACTION:* the action that was applied to parent to generate the node
- *n.PATH-COST:* the cost, denoted by *g(n)*, of the path from initial state to node



Implementation: states vs. nodes

innovate achieve lead

- Nodes vs States
  - Node – a bookkeeping data structure used to represent the search tree
  - State – configuration of the world
  - Two different nodes might have same state – if that state is generated from two different paths

- Frontier – needs to be stored in a queue with a strategy to help pick the next node for expansion

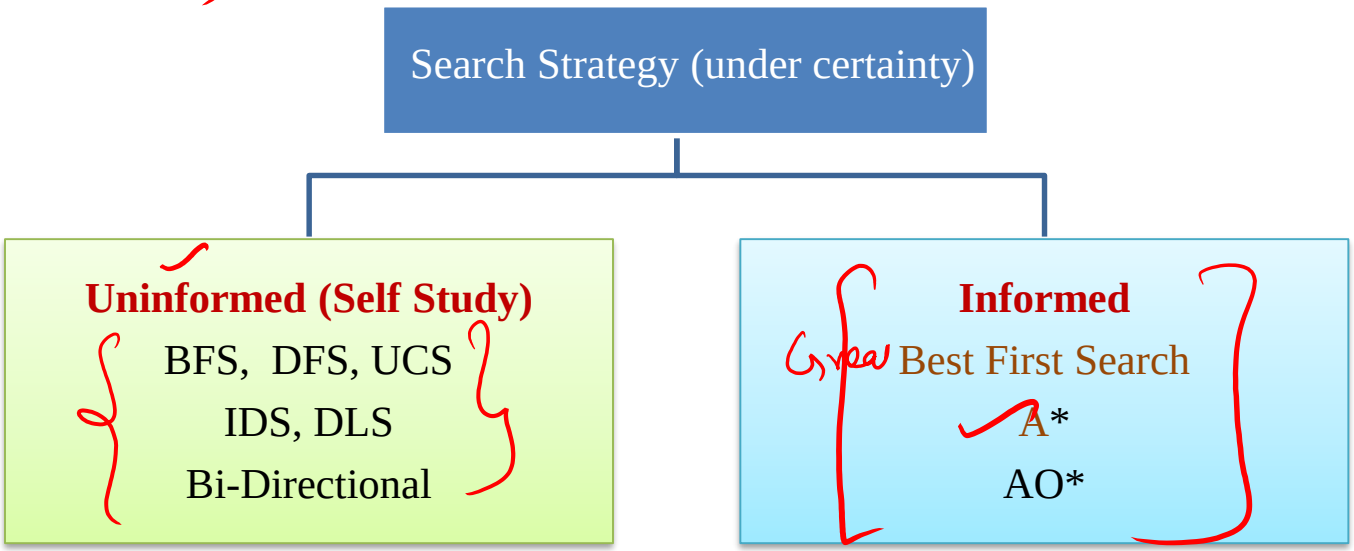
- Strategies can be
  - FIFO (First In First Out) – Pops the oldest element of the queue
  - LIFO (Last In First Out) – Pops the newest element of the queue
  - Priority Queue – pops the element with highest priority

Search Strategies

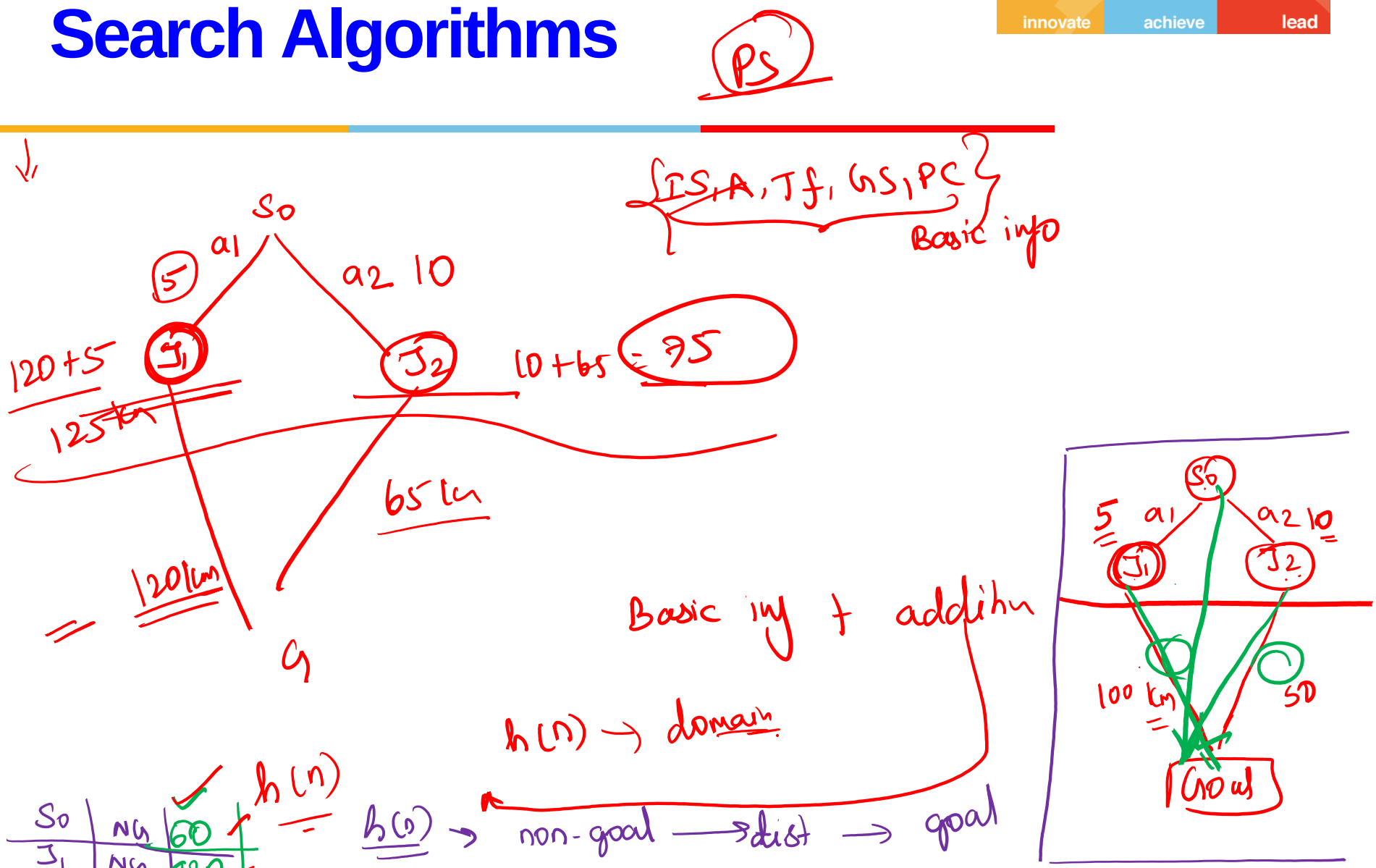
- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
  - systematicity: does it visit each search node at most once?
- Time and space complexity are measured in terms of
  - b: maximum branching factor of the search tree
  - *d:* depth of the least-cost solution (of the shallowest goal, i.e., number of steps from initial node)
  - m: maximum depth of the state space (may be ∞)

Types of Search Strategies

- Choosing the current state, testing possible successor function, expanding current state to generate new state is called Traversal. Choice of which state to expand – Search Strategy
- Broadly, there are two categories: (a) Uninformed Search and (b) Informed search strategies.



Search Algorithms



BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

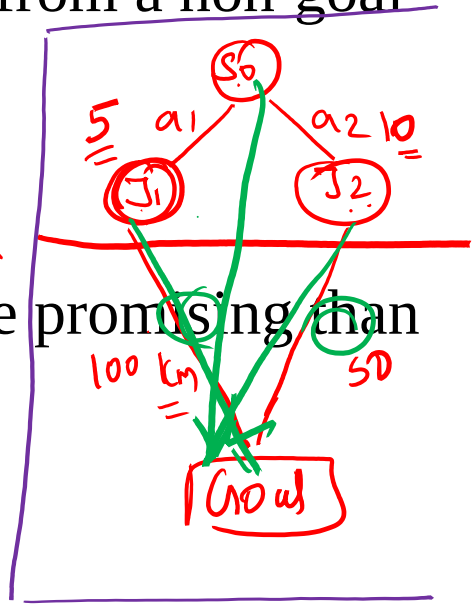
Search Algorithms

Uninformed Search Algorithms

- No additional information about states beyond what is provided in problem formulation. They use only information available in the problem definition.
- Generate successors and distinguish a goal state from a non-goal state

Informed Search Algorithms

- Strategies that know if one non-goal state is more promising than another non-goal state
- Also called Heuristic Search strategies



Uninformed Search Algorithms

1. Breadth First Search
2. Uniform Cost Search
3. Depth First Search
4. Depth Limited Search
5. Iterative deepening Depth First Search
6. Bi-directional Search

Watch the animations of each and play around these searches.
These are not in the syllabus but are pre-requisites.

Problem

- All these methods are slow (blind)

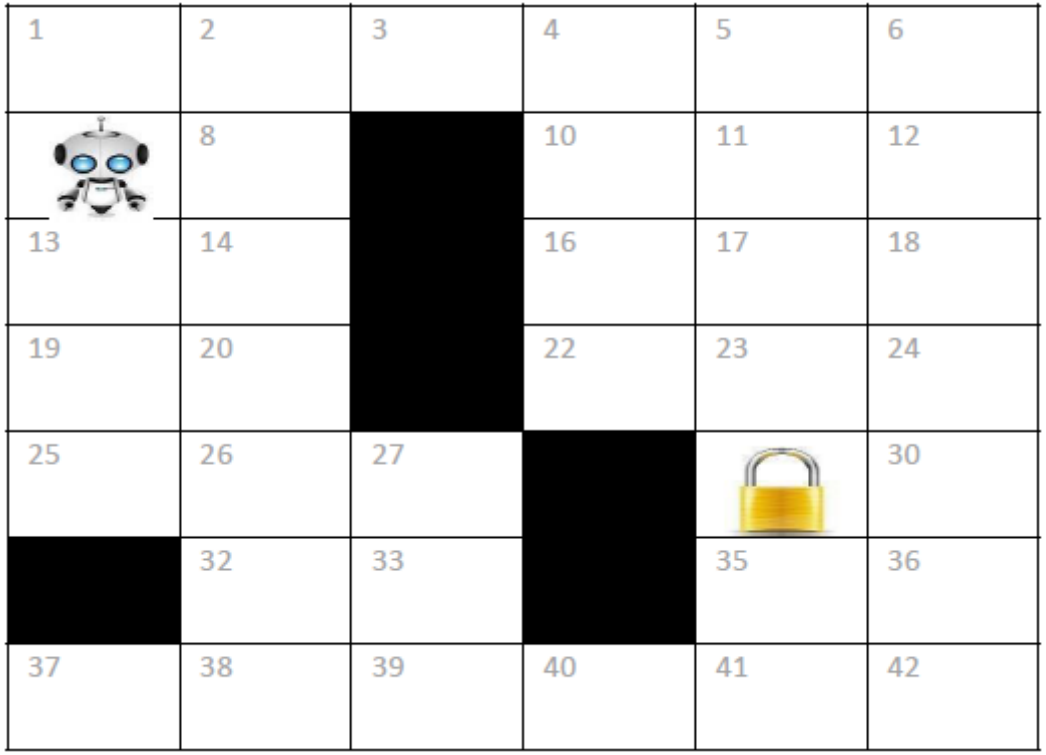
Solution → add guidance ("heuristic estimate")
→ "informed search"

Exercise 1: Write the PEAS for an Interactive English tutor such as Grammarly

- Performance measure: ?
- Environment: ?
- Actuators: ?
- Sensors: ?

Exercise 2: Path Finding Robot

PEAS for this problem?



Observability?

Number of Agents?

Determinism?

Dynamicity?

Episodic or Sequential?

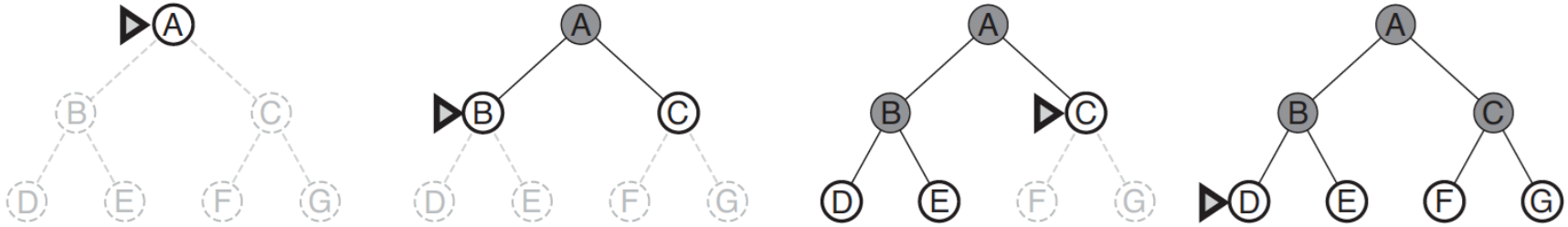
No of States?

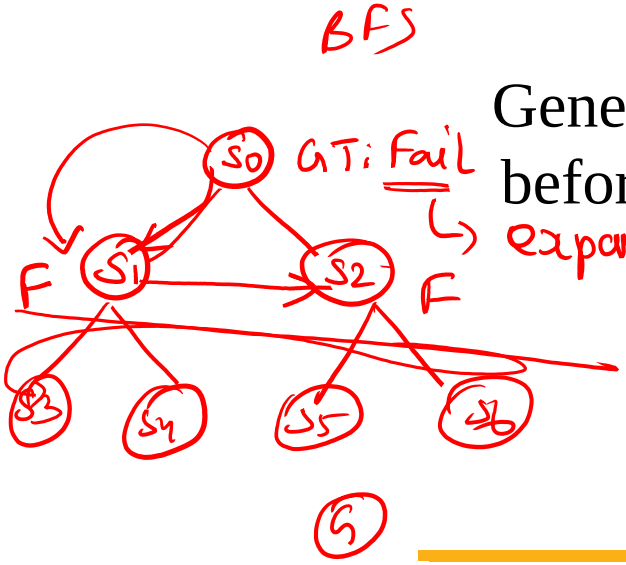
Exercise 3: Identify the Environment

- 15-Puzzle game
- Poker
- Medical Diagnosis
- Taxi Driving

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Breadth First Search (BFS) Summary

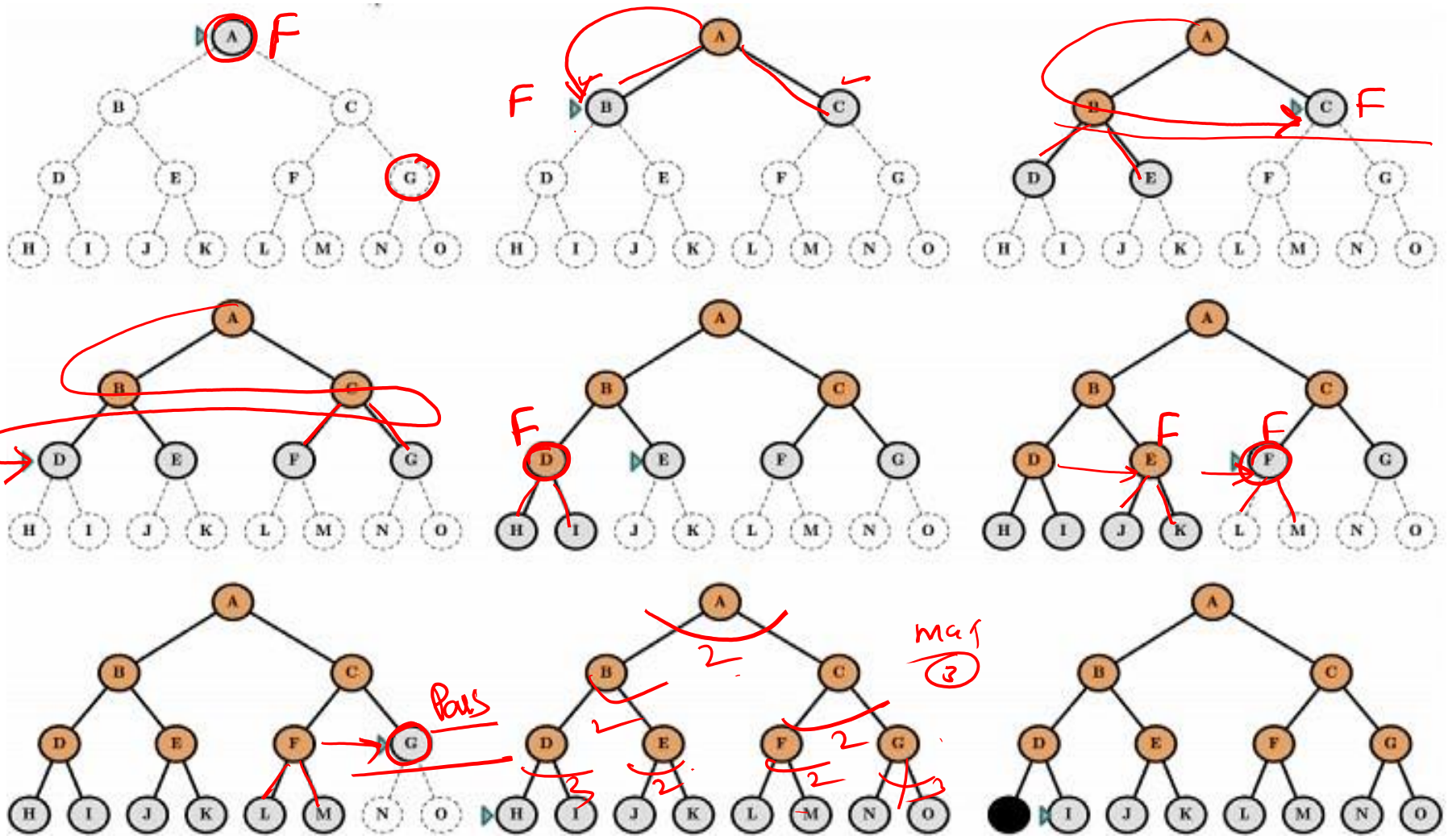




Generate all nodes at a given depth before proceeding to deeper nodes

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Breadth First Search (BFS) Summary



Breadth First Search (BFS)

Complete – If the shallowest goal node is at a depth d, BFS will eventually find it by generating all shallower nodes

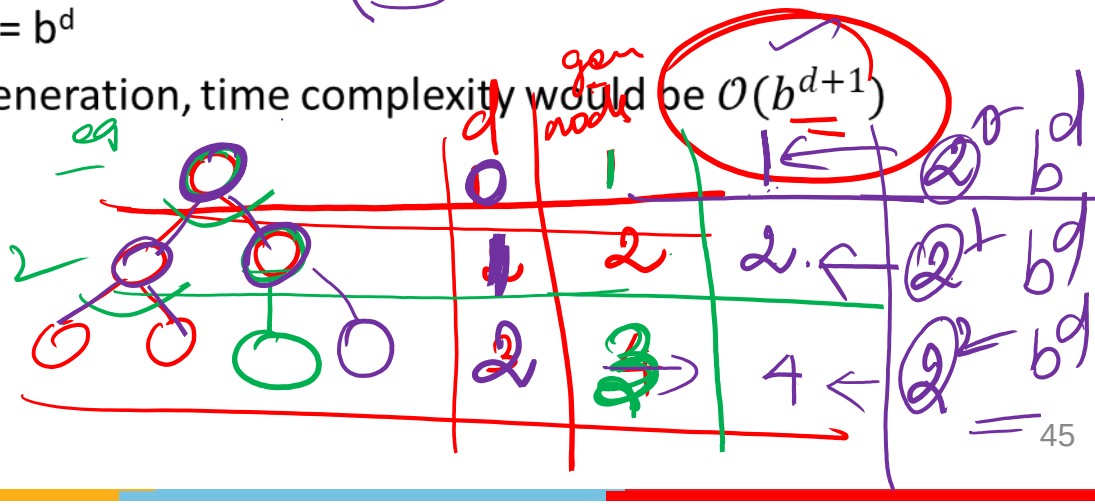
Optimal – Not necessarily. Optimal if path cost is non-decreasing function of depth of node. E.g., all actions have same cost

Time Complexity – $O(b^d)$ b - branching factor, d – depth

- Nodes expanded at depth 1 = b
- Nodes expanded at depth 2 = b^2
- Nodes expanded at depth d = b^d
- Goal test is applied during generation, time complexity would be $O(b^{d+1})$

Space Complexity – $O(b^d)$

- $O(b^{d-1})$ in explored set
- $O(b^d)$ in frontier set



Breadth First Search (BFS)

Complete – If the shallowest goal node is at a depth d, BFS will eventually find it by generating all shallower nodes

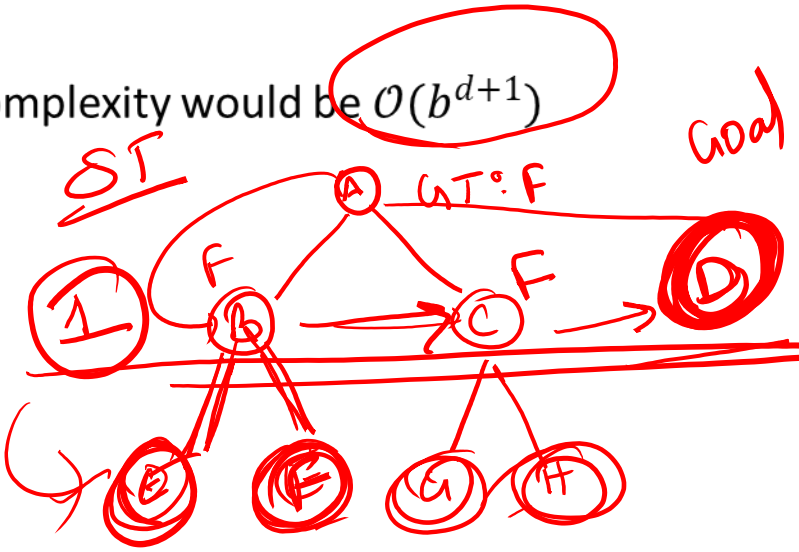
Optimal – Not necessarily. Optimal if path cost is non-decreasing function of depth of node. E.g., all actions have same cost

Time Complexity – $\mathcal{O}(b^d)$ b - branching factor, d – depth

- Nodes expanded at depth 1 = b
- Nodes expanded at depth 2 = b^2
- Nodes expanded at depth d = b^d
- Goal test is applied during generation, time complexity would be $\mathcal{O}(b^{d+1})$

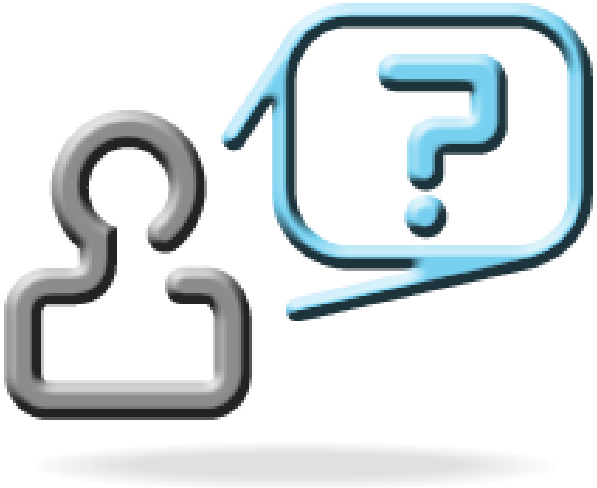
Space Complexity – $\mathcal{O}(b^d)$

- $\mathcal{O}(b^{d-1})$ in explored set
- $\mathcal{O}(b^d)$ in frontier set



generated

dH



We will explore more on Search techniques in the next class …

Thank You for your time & attention !

Slides are Licensed Under : CC BY-NC-SA 4.0